Final Portfolio

Literature Review: The Boyer-Moore String Search Algorithm

Jerod Sun
12/15/2019
WRIT-015
Professor Lipscomb

Start typing the name of a person you want to text on your phone. Search "cat gifs" in Google. Find the word "perspicuity" in a speech. End users today expect nearly instantaneous results from these interactions – interactions that depend on the ability of machines to process text and images at nearly-instantaneous speeds. The fundamental principles behind these interactions are string pattern matching algorithms. One major breakthrough came in 1977, when Robert S. Boyer & J Strother Moore, professors at UT Austin, invented the namesake Boyer-Moore algorithm. Boyer-Moore is a string search algorithm that utilizes optimizations to reduce the number of computations it takes to find a substring within a larger block of text. (Boyer) This literature review explores the motivating factors behind developing Boyer-Moore and the impact it continues to have in the field of computer science.

Computer science deals with optimizing the theoretical computational complexity of algorithms in practice. Historically, the number of computations per second needed for practical string pattern matching applications greatly exceeded the processing capacity of microprocessors. Before Boyer-Moore, the most common way of implementing a string search function was to use a “brute-force” algorithm. Brute-force takes O(mn) time complexity, rendering it impractical for all but the smallest datasets or most trivial tasks. Computer scientists constantly try to find more efficient algorithms than brute-force approaches; indeed, the bulk of modern technological innovation has come from successful algorithmic discoveries. A second motivating factor was the hardware limitations of the day. In 1965, Intel’s co-founder, Gordon E. Moore (no relation to J Strother Moore), postulated that microchips can double their transistors every two years (Moore). This principle, known as "Moore's Law", means that, in the 42 years from 1977 to 2019, current processors are faster than those 42 years ago by a magnitude of 2^21. That’s more than two million times slower in 1977!

Boyer, Moore and other computer scientists needed to work within the limitations of the processing speeds of 1977. However, the most common implementation of string searches relied on brute-force. When searching for a pattern in a string of characters, the "brute-force" method is to simply try at each index until the program asserts a mismatch or a complete match. This, at worst, means that a string of length *n* and pattern of length *m* will require *n* times *m* comparisons. Professors Boyer and Moore came up with two heuristics that drastically reduced the number of comparisons to be made. These heuristics are referred to as the bad character heuristic and the good suffix heuristic, also known as the Character Jump heuristic and the Looking Glass Heuristic. The Looking Glass heuristic starts the matching process from the end of the pattern, instead of the front. By starting comparisons at the suffix, a mismatch can immediately skip impossible matches. A practical demonstration is to match the phrase "A short guide for teachers" in a book that contains "A short guide for students" before the phrase "A short guide for teachers." When Boyer-Moore reaches the first string, it considers the last character: s. After asserting a match between the two patterns, Boyer-Moore next considers t and r, which do not match. As we are starting from the first possible match, if there was a match, it would all be matched. But all the characters in the previous part of the string - which we did not look at - could not possibly be part of the match. Therefore, there is no need to look at the previous characters at all. Then comes the Character Jump heuristic: the whole index can be shifted forward. Boyer-Moore refers to a "lookup table" to maximize the count it can skip ahead at each "successfully failed" match. In contrast, if an algorithm matches from the beginning of the string, it will not have this forward-looking advantage, and it still has to make an initial comparison at each stage. These optimizations are especially pertinent to the English language, where the length of the pattern logarithmically decreases the number of comparisons to be made in

practice. According to Boyer, “For an English pattern of length 5, the algorithm typically inspects 0.24 characters for every character passed.” (Boyer, 766)

Boyer-Moore was part of a paradigm shift in algorithmic analysis. Before Boyer-Moore, most algorithmic analyses focused on improvements to the "worst-case" time. Professors Boyer and Moore took a different approach: what are possible heuristics that improve the expected time, instead of the worst-case time? This line of reasoning persuaded other researchers to approach algorithmic analysis from the angle of optimizing the average-case time complexity. While understanding worst-case time complexity is necessary to make justifications and proofs, heuristics are much more useful in practice. In doing so, heuristics necessarily sacrifice something computationally. But in this case, the "something" is irrelevant: imperfect and confirmed mismatches will not be processed and will simply be skipped. Because the optimizations consist entirely of heuristics, Boyer-Moore has a theoretical time complexity of O(mn) – the same as a brute-force approach. While the algorithmic bound is quadratic, in the vast majority of scenarios, Boyer-Moore is faster than other standard algorithms in English text matching (Goodrich).

In the same year, 1979, researchers Knuth, Morris and Pratt independently developed the KMP algorithm (Knuth). KMP replaces the Looking Glass heuristic by traversing the string left-to-right, instead of from the suffix. The heuristic KMP uses is a “character jump” heuristic. This distinct heuristic determines where a next match can begin after each mismatch, eliminating the need to double-check previously matched characters from a different index. By using an algorithmic proof dependent on separating words as individual matches, this has a worst-case time complexity of linear time with a high constant factor, compared to Boyer-Moore’s quadratic time. But in practice, Boyer-Moore is generally faster on English language texts. A car driving

analogy describes Boyer-Moore relative to KMP: one day a year, the freeway will have an accident, and you will get home faster by taking the local route - which will always take the same amount of time - but, on the other 364 days, the freeway will be faster.

While Boyer-Moore is usually faster, there are several scenarios where Boyer-Moore tends toward the worst-case time. Due to the nature of the lookup table, Boyer-Moore tends to be inefficient on texts with small alphabets, such as DNA sequences. By applying Boyer-Moore to a sequence with only a single character mismatch, Boyer-Moore has to iterate backwards O(mn) times. With large alphabets, diversified texts, and longer patterns to match, Boyer-Moore is more efficient. In contrast, the heuristics utilized by KMP are more efficient with small alphabets, constant repeats and runs of similar sequences. However, recent improvements in Boyer-Moore have allowed it to be more efficient on these types of strings, driven by practical applications in other fields. In biology, a common task is to match DNA sequences. DNA sequences have 4 bases as characters when converted to text. A specialized application of string-matching can be optimized for only 4 characters, keeping in mind that there are certain unique patterns that make it different from natural language texts. Next Generation Sequencing is a recent development that has allowed genetic sequencing to be conducted in days instead of years, greatly expanding the need for DNA pattern matching. Boyer-Moore forms the foundation for several new algorithms optimized for DNA searches. One of these was proposed in 2014 by Nadia Ben Nsira, Thierry Lecroq and Mourad Elloumi in their paper “A fast pattern matching algorithm for highly similar sequences”. (Nsira) Their variant extended Boyer-Moore to exhibit the best performance in practice for this highly specialized application.

While Boyer-Moore, by itself, has a quadratic worst-case time complexity, other computer scientists have discovered several new algorithms deriving from Boyer-Moore with

elaborate proofs for stricter theoretical bounds on computational complexity. In 1979, two years after the discovery, Zvi Galil proved theoretical linear time complexity in Boyer-Moore with a slight optimization, known as the "Galil Rule" (Galil). This optimization adds “memory” to the algorithm, allowing periodic repeats to be saved and skipped in repeat searches on the next index. Building on this work, Professors Shmuel T. Klein and Miri Kopel Ben-Nissan developed pre-computed tables to process entire blocks such as bytes or words in their article “Accelerating Boyer Moore Searches on Binary Texts” (Klein). This is similar to the process in the KMP algorithm – indeed, these algorithms are complementary approaches to the same problem. The interconnectedness between these two algorithms is why they are often presented together in academic literature. Together, these algorithms revolutionized everything that depends on efficient string searches, drastically shortening one of the bottlenecks in academic research and practical application.

Along with KMP, Boyer-Moore is included in the standard libraries of many programming languages. It is the primary implementation in one of the most famous UNIX command line utilities, grep (Chou). However, the heuristics introduced by Boyer-Moore and KMP have influenced fields far beyond string matching. At the lowest level, applications of Natural Language Processing and advanced linguistic research in computer science rely on Boyer-Moore. Due to the nature of data collection, datasets often contain abbreviations, misspellings or mistakes. Such “Dirty data” created a need for algorithms that can give the probabilities for the chance of a match, either when no perfect match appears, or to include all potential true matches with inaccuracies in data collection. These “fuzzy matching” applications include plagiarism detection algorithms. Plagiarism detection relies on computing the similarities of strings, expanded to factors like synonyms, word order, and misspellings. One type of

plagiarism detection algorithm relies on a “fast parameterized Boyer-Moore algorithm using q-gram.” (Pandey, 150) In addition, applications of fuzzy matching frequently inspire new methods in Machine Learning. Classical Machine Learning algorithms all implement different sorts of heuristics to save computational time. (Pandey, 341)

Today, Boyer-Moore is standard literature in any introductory algorithmic book. It presents a straightforward optimization for what, at first glance, does not appear to have a better solution. Understanding why an algorithm works is critical in deciding which algorithm to apply to a problem and to make further improvements on the algorithm itself. While efficient string search algorithms are continuously being developed, almost all of them trace their inspiration back to Boyer-Moore or KMP. Next time you see instantaneous results as you type on your phone, think of the journey it took to reach this stage, starting from this discovery by Professors Boyer and Moore.

Bibliography:

Boyer, Robert S. & Moore, J Strother (1977). "A Fast String Searching Algorithm". Communications of the ACM. New York: Association for Computing Machinery. 20(10), 762-772.

Chou, Daniel. Personal Interview. November 8, 2019. Georgetown University.

Galil, Z. (1979). “On improving the worst case running time of the Boyer-Moore string matching algorithm”. Communications of the ACM. New York: Association for Computing Machinery. 22(9), 505-508.

Klein, S.T., Kopel Ben-Nissan, Miri. "Accelerating Boyer Moore searches on binary texts" Proc. Conference On Implementation And Application of Automata, LNCS 4783, 2007, 130-143.

Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser. 2013. Data Structures and Algorithms in Python (1st ed.). Wiley Publishing.

Moore, Gordon E. (1998). "Cramming more components onto integrated circuits". Proceedings of the IEEE. 86(01), https://www.cs.utexas.edu/~fussell/courses/cs352h/papers/moore.pdf. Accessed 10 November, 2019.

Knuth, Donald E., Morris, James H. and Pratt, Vaughan R (1977). "Fast Pattern Matching in Strings." *SIAM Journal on Computing* 6(2), 323-350.

Nsira, Nadia & Lecroq, Thierry & Elloumi, Mourad. (2015). "A fast Boyer-Moore type pattern matching algorithm for highly similar sequences." International Journal of Data Mining and Bioinformatics. 13(3), 266-288.

Pandey K.L., Agarwal S., Misra S., Prasad R. (2012) Plagiarism Detection in Software Using Efficient String Matching. Computational Science and Its Applications, ICCSA 2012. Lecture Notes in Computer Science, vol 7336. Springer, Berlin, Heidelberg.